부록: 자연어 데이터 처리

모듈 - 2

강사: 장순용 박사

순서

2. 자연어 데이터 처리:

자연어 분석 (NLP)

자연어 분석이란?

- 언어/텍스트 데이터에서 패턴을 추출하여 분류, 군집화, 요약, 감성분석, 등을 행하는 것.
- 언어학과 인공지능의 교차점에 있다.
- **통계 & 인공지능** 모형에 기반한다. ⇐ 인간이 언어를 이해하는 방식과는 다르다.
- 데이터의 구조화 과정이 포함된다.

말뭉치

말뭉치 (Corpus):

- 연구 재료로서 언어적 자료의 집합을 의미한다.

  ⇒ 작게는 소설 한편이 될 수 있다.

  ⇒ 크게는 수십억 어절 이상으로 이루어진 텍스트 자료의 모음.

- **원시 말뭉치** (Raw Corpus): 텍스트를 컴퓨터로 읽을 수 있는 자료로 만들어서 저장해 놓은 것.
- **가공된 말뭉치** (Tagged Corpus): 수집된 텍스트 데이터를 형태소 분석이나 어휘, 품사 정보, 문헌, 내용 등으로 분류할 수 있도록 인공적으로 가공해 놓은 말뭉치.

전처리

전처리 과정 (Pre-processing):

1). 분절 (Tokenization):

⇒ 원 텍스트를 단어 또는 문장 단위로 분리해 놓는 것.

2). 정제 (Cleaning):

⇒ 분리된 문장에서 문장 부호, 특수 문자, 숫자 등 불필요한 요소 제거. (정규 표현식 활용)

⇒ 과하게 짧거나 단모음, 단자음 제거. (정규 표현식 활용)

3). 정규화 (Normalization):

⇒ 영문인 경우에는 소문자화.

⇒ 불용어 (Stop words) 제거.

⇒ 어간추출 (Stemming), 원형복원 (Lemmatization).

전처리 : 분절

분절 (Tokenization):

- 일반적으로 데이터 전처리의 첫 단계.

a). 문장 단위 분절:

⇒ 보통 문장의 끝은 점 (마침표), 물음표, 느낌표 등으로 알 수 있다.

⇒ 하지만 점이 문장의 끝을 나타내지는 않는 경우도 있다.

예). "He got his *M.D.* from the university of Wisconsin."

⇒ 이러한 예외상황도 인지하는 분절이 필요하다.

b). 단어 단위 분절:

⇒ 보통 스페이스 또는 쉼표 등 문장 부호로 단어를 구별 지을 수 있다.

⇒ 하지만 영문의 경우 어포스트로피 (Apostrophe) 사용은 분절을 어렵게 만들 수 있다.

예). "*Don't* lose your hope, *everything's* possible."

전처리 : 정규화

정규화: 불용어와 키워드.

- 특별한 정보를 제공하지 못하는 단어를 "**불용어**"(Stop words)라 부른다.

  예). 영문에서의 관사: "the", "a", "an".

  예). 영문에서의 전치사: "on", "with", "into", "upon", etc.

- 불용어는 "데이터 정규화" 과정에서 처리하게 된다. ⇒ 불용어 사전을 사용해서 제거한다.
- 불용어의 반대는 "**가용어**"이다.
- 가용어 중에서 문서의 주체어를 "**키워드**"라고 부른다.

  ⇒ 보통은 문서 내에서 **발생 빈도가 높은** 단어들을 키워드로 선정한다.

  ⇒ 키워드 선정은 분석하고자 하는 목적 및 문서의 특성을 따르는 것이 원칙이다.

전처리 : 정규화

정규화: 불용어 (영문).

- NLTK 라이브러리의 영문 불용어 사전에는 179 단어가 있다:

i, me, my, myself, we, our, ours, ourselves, you, you're, you've, you'll, you'd, your, yours, yourself, yourselves, he, him, his, himself, she, she's, her, hers, herself, it, it's, its, itself, they, them, their, theirs, themselves, what, which, who, whom, this, that, that'll, these, those, am, is, are, was, were, be, been, being, have, has, had, having, do, does, did, doing, a, an, the, and, but, if, or, because, as, until, while, of, at, by, for, with, about, against, between, into, through, during, before, after, above, below, to, from, up, down, in, out, on, off, over, under, again, further, then, once, here, there, when, where, why, how, all, any, both, each, few, more, most, other, some, such, no, nor, not, only, own, same, so, than, too, very, s, t, can, will, just, don, don't, should, should've, now, d, ll, m, o, re, ve, y, ain, aren, aren't, couldn, couldn't, didn, didn't, doesn, doesn't, hadn, hadn't, hasn, hasn't, haven, haven't, isn, isn't, ma, mightn, mightn't, mustn, mustn't, needn, needn't, shan, shan't, shouldn, shouldn't, wasn, wasn't, weren, weren't, won, won't, wouldn, wouldn't

전처리 : 정규화

정규화: 어간추출 & 원형복원.

- 어간을 분리해내는 텍스트 처리기술로 Stemming, Lemmatization 등이 있다.
- Stemming (어간추출):

  ⇒ 단어의 접미사나 어미를 제거해 주어서 형태를 일치시킨다.

  예). “cried” → “cri”

  예). “remembrance” → “remembr”

- Lemmatization (원형복원):

  ⇒ 단어를 사전적 의미가 있는 근본적 형태로 변환한다.

  예). “wolves” → “wolf”

형태소 분석

형태소 분석:

- 형태소란 의미가 있는 최소의 단위로서 더 이상 분리가 불가능한 요소이다.

  ⇒ 문법적, 관계적 뜻을 나타내는 단어 또는 단어의 부분이다.

  ⇒ 어간과 어미 단위도 형태소로 간주한다.

- 형태소 분석이란 주어진 **어절** 또는 **단어**의 형태소에서 **기본형** 및 **품사 정보**를 추출하는 것.

  ⇒ **어절**: 한국어에서는 띄어쓰기를 기준으로 어절을 구분한다.

  예). “나는 사과를 먹는다” = “나는” + “사과를” + “먹는다”.

  ⇒ **단어**: 어절을 구성하는 단위로 어절은 하나 또는 두 개 이상의 단어로 구성됨.

  예). “나는” = “나” + “는”.

형태소 분석

형태소 분석: 품사 태깅 (Part of Speech Tagging, POS).

- 단어의 모호성을 제거하는 품사 태깅:

⇒ 영문의 경우 “Penn Treebank tag set”에 의한 품사 태깅.

예). NN 명사 : desk

PRP 인칭대명사 : I, she, he

VB 동사 원형: take

⇒ 국문의 경우 “21세기 세종계획 품사 태그 셋”, “KAIST 품사 태그 셋” 등이 있다.

형태소 분석

형태소 분석: 품사 태깅 (Part of Speech Tagging, POS).

- 영문 Penn Treebank tag set:

CC	coordinating conjunction	
CD	cardinal digit	
DT	determiner	
EX	existential there (like: "there is" ... think of it like "there exists")	
FW	foreign word	
IN	preposition/subordinating conjunction	
JJ	adjective	'big'
JJR	adjective, comparative	'bigger'
JJS	adjective, superlative	'biggest'
LS	list marker	1)
MD	modal	could, will
NN	noun, singular	'desk‘
NNS	noun plural	'desks'
NNP	proper noun, singular	'Harrison'
NNPS	proper noun, plural	'Americans'
PDT	predeterminer	'all the kids'
POS	possessive ending	parent's

형태소 분석

형태소 분석: 품사 태깅 (Part of Speech Tagging, POS).

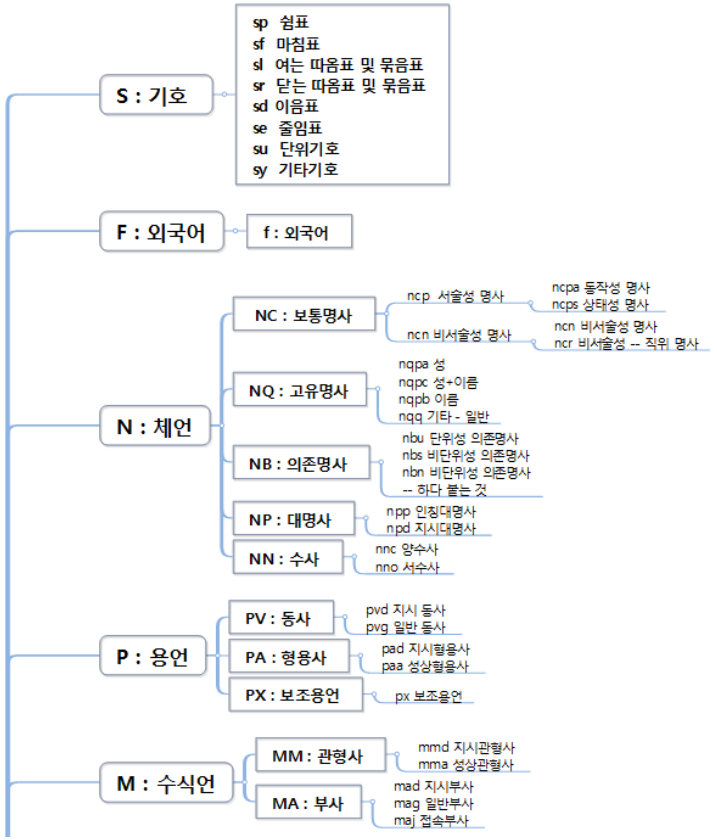
- 영문 Penn Treebank tag set:

PRP	personal pronoun	I, he, she	
PRP$	possessive pronoun	my, his, hers	
RB	adverb	very, silently,	
RBR	adverb, comparative	better	
RBS	adverb, superlative	best	
RP	particle	give up	
TO	to	go 'to' the store.	
UH	interjection	errrrrrrrm	
VB	verb, base form	take	
VBD	verb, past tense	took	
VBG	verb, gerund/present participle		taking
VBN	verb, past participle	taken	
VBP	verb, sing. present, non-3d		take
VBZ	verb, 3rd person sing. present		takes
WDT	wh-determiner	which	
WP	wh-pronoun	who, what	
WP$	possessive wh-pronoun	whose	
WRB	wh-abverb	where, when	

형태소 분석

형태소 분석: 품사 태깅 (Part of Speech Tagging, POS).

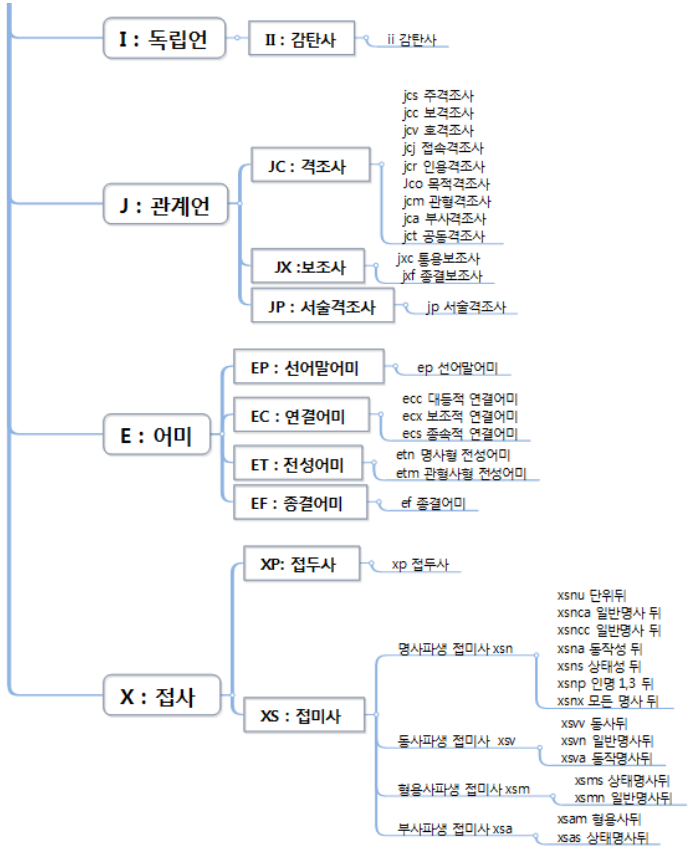
- 국문 KAIST 품사 태그 셋:



형태소 분석

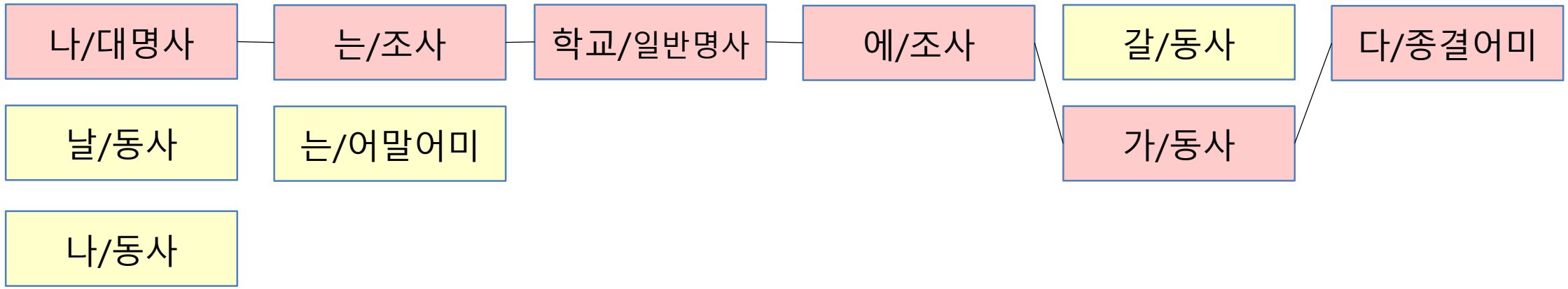
형태소 분석: 품사 태깅 (Part of Speech Tagging, POS).

- 국문 KAIST 품사 태그 셋:



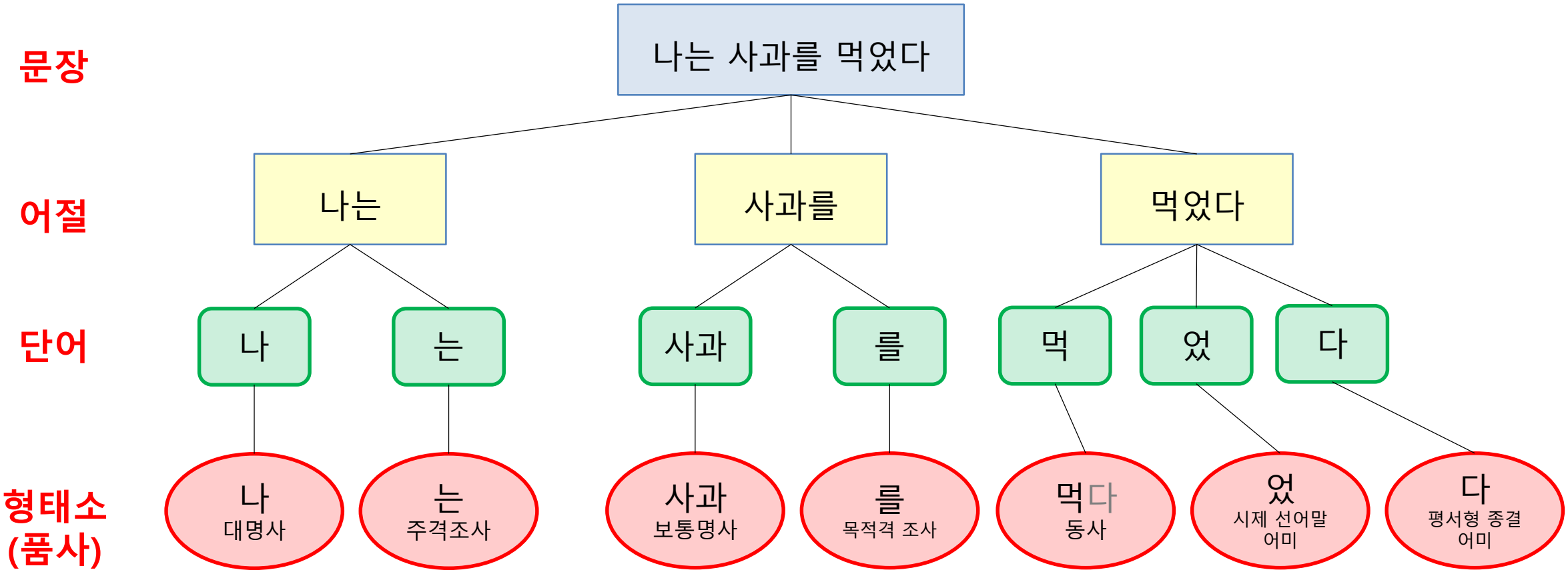
형태소 분석

형태소 분석: 품사 태깅 (Part of Speech Tagging, POS).



형태소 분석

형태소 분석: 도식화.



NLTK 라이브러리 소개

NLTK (Natural Language Toolkit) 라이브러리:

- 영문 (+유럽계 언어) 자연어 처리 및 분석용 파이썬 라이브러리.

  a). 분절 (Tokenization): sent_tokenize, word_tokenize, etc. 다음과 같이 관련 데이터를 내려 받아야 한다.

  nltk.download('punkt')

  b). 불용어: 총 179 개 (영문). 다음과 같이 관련 데이터를 내려 받아야 한다.

  nltk.download('stopwords')

  c). 어간추출 (Stemming): PorterStemmer, LancasterStemmer, SnowballStemmer, etc.

  d). 원형복원 (Lemmatization): WordNetLemmatizer.

  e). 품사 태깅 (POS Tagging): Penn Treebank tag set 사용.

  f). 감성 분석을 위한 리소스: WordNet, SentiWordNet, etc.

KoNLPy 라이브러리 소개

KoNLPy 라이브러리:

- 국문 자연어 처리를 위한 라이브러리.
- 다양한 한글 형태소 분석기 제공: Kkma, Hannanum, Komoran, Mecab, Okt, Twitter.

  ⇒ 형태소 단위로의 분절 기능.

  ⇒ 품사 태깅 기능.

- Windows에서는 설치 절차가 다소 복잡하다.

1). Java SDK 다운로드 페이지에서 JDK를 다운로드 하고 설치한다.

2). JAVA의 JDK 폴더로의 경로를 환경 변수로 설정 한다 (JAVA_HOME).

3). Jype1 패키지 설치 (*https://www.lfd.uci.edu/~gohlke/pythonlibs/#jpype*)

4). KoNLPy 패키지 설치.

순서

2. 자연어 데이터 처리:

2.1. 자연어 분석 (NLP) 개요.

2.2. 정규 표현식과 전처리.

문자열 처리

문자열 처리에 유용한 파이썬 함수 또는 메서드:

함수/메서드	설명
x.lstrip()	문자열 x에서 왼쪽 스페이스 제거.
x.rstrip()	문자열 x에서 오른쪽 스페이스 제거.
x.strip()	문자열 x에서 양쪽 스페이스 제거.
x.replace(str1, str2)	문자열 x에서 str1를 str2로 대체.
x.count(str)	문자열 x에서 str의 발생 횟수.
x.find(str)	문자열 x에서 str의 위치. 없으면 -1 반환.
x.index(str)	문자열 x에서 str의 위치. 없으면 오류 발생.
y.join(str_list)	리스트 str_list의 원소들을 y로 연결.
x.split(y)	y 를 separator로 사용하여 문자열 x을 자른다.
x.upper()	문자열 x를 대문자화.
x.lower()	문자열 x를 소문자화.
len(x)	문자열 x의 길이.

문자열 처리

정규 표현식:

- 문자열 패턴을 만드는 목적으로 사용된다.
- 복잡한 문자열 인식과 처리의 목적으로 사용된다.

  ⇒ 자연어 데이터의 전처리 (정제) 용도로 유용하다.
- 기존의 문자열 함수의 조합보다 함축적이고 강력하다.
- 파이썬 이외에도 많은 프로그래밍 언어에서 지원된다.

정규 표현식

메타문자 (Meta Characters):

- 정규 표현식에서 특별한 의미를 갖는 문자이다.

. ^ $ * + ? \ | { } [ ] ( )

- 문자열 패턴을 만드는 목적으로 사용된다.

정규 표현식

메타문자: [ ]

- 문자 클래스를 만들어 준다. [와] 사이에는 어떠한 문자도 들어갈 수 있다.
- “[와] 사이의 문자들과 매치”라는 의미를 갖는다.
- “[abc]”라는 정규표현식의 의미는 “a 또는 b 또는 c 중 일치”이다.

정규표현식	문자열	Match?	설명
“[abc]”	“a”	Yes	“a”, “b”, “c” 과 일치하는 문자인 “a”가 포함되어 있음.
“[abc]”	“before”	Yes	“a”, “b”, “c” 과 일치하는 문자인 “b”가 포함되어 있음.
“[abc]”	“dude”	No	“a”, “b”, “c” 중 어느 하나도 포함되어 있지 않음.

정규 표현식

메타문자: [ ]

- [ ] 안의 두 문자 사이에 하이픈(-)을 사용하게 되면 두 문자 사이의 범위를 의미.

  예). “[a-c]”는 “[abc]”와 동일하다.

  예). “[0-5]”는 “[012345]”와 동일하다.

  예). “[a-zA-Z]” : 알파벳 전체.

  예). “[0-9]” : 숫자.

정규 표현식

메타문자: [^]

- “[^와] 사이의 문자들 이외의 것과 일치 함”을 의미.
- “[^abc]”라는 정규표현식의 의미는 “a 아니고, b 아니고, c도 아닌 문자”이다.

정규표현식	문자열	Match?	설명
“[^abc]”	“a”	No	“a”, “b”, “c” 이외의 문자 없음
“[^abc]”	“before”	Yes	“a”, “b”, “c” 이외의 문자 있음.
“[^abc]”	“dude”	Yes	“a”, “b”, “c” 이외의 문자 있음.

정규 표현식

메타문자: [ ] 와 [^]

- 다음과 같은 **단축 표현**이 있다.

→ “\w” = “[a-zA-Z0-9_]”

→ “\W” = “[^a-zA-Z0-9_]”

→ “\d” = “[0-9]”

→ “\D” = “[^0-9]”

→ “\s” = 스페이스.

→ “\S” = 스페이스 아님.

정규 표현식

메타문자: 점 .

- 정규 표현식의 점 “.” 메타문자는 모든 문자와 매치됨.
- Placeholder의 역할을 함.
- 메타문자가 아닌 점은 “\.” 또는 “[.]”로 나타낸다.

정규표현식	문자열	Match?	설명
“a.b”	“aab”	Yes	가운데 문자 “a”가 모든 문자를 의미하는 .과 일치.
“a.b”	“a0b”	Yes	가운데 문자 “0”가 모든 문자를 의미하는 .과 일치.
“a.b”	“abc”	No	“a”문자와 “b”문자 사이에 어떠한 문자도 없음.

정규 표현식

메타문자: *

- “*”의 위치에서 바로 앞에 있는 문자가 0회 부터 무한대의 횟수까지 반복될 수 있다는 의미.

정규표현식	문자열	Match?	설명
“ca*t”	“ct”	Yes	“*”의 위치에서 “a”가 0회 출현.
“ca*t”	“cat”	Yes	“*”의 위치에서 “a”가 1회 출현.
“ca*t”	“caaat”	Yes	“*”의 위치에서 “a”가 3회 반복.

정규 표현식

메타문자: +

- “+”의 위치에서 바로 앞에 있는 문자가 최소 1회 이상 반복.

정규표현식	문자열	Match?	설명
“ca+t”	“ct”	No	“+”의 위치에서 “a”가 0회 출현.
“ca+t”	“cat”	Yes	“+”의 위치에서 “a”가 1회 출현.
“ca+t”	“caaat”	Yes	“+”의 위치에서 “a”가 3회 반복.

정규 표현식

메타문자: ?

- “?”의 위치에서 바로 앞에 있는 문자가 0회 또는 1회 출현.

정규표현식	문자열	Match?	설명
“ca?t”	“ct”	Yes	“?”의 위치에서 “a”가 0회 출현.
“ca?t”	“cat”	Yes	“?”의 위치에서 “a”가 1회 출현.
“ca?t”	“caat”	No	“?”의 위치에서 “a”가 2회 반복.

정규 표현식

메타문자: {*m*}

- 바로 앞에 있는 문자가 정확하게 ***m***회 반복된다는 의미.

정규표현식	문자열	Match?	설명
“ca{2}t”	“ct”	No	“a”가 2회 반복되지 않음.
“ca{2}t”	“cat”	No	“a”가 2회 반복되지 않음.
“ca{2}t”	“caat”	Yes	“a”가 정확하게 2회 반복됨. 매치!

정규 표현식

메타문자: {*m*,*n*}

- 바로 앞에 있는 문자가 ***m***회에서 ***n***회 사이 반복된다는 의미.

정규표현식	문자열	Match?	설명
"ca{2,5}t"	"cat"	No	"a"가 1회 반복됨.
"ca{2,5}t"	"caat"	Yes	"a"가 2회 반복됨. (2와 5 사이!)
"ca{2,5}t"	"caaaaaat"	No	"a"가 6회 반복됨.

정규 표현식

메타문자: ^

- 문자열의 제일 앞 부분과 일치함을 의미.
- “[^]”과는 다르니 주의한다.

정규표현식	문자열	Match?	설명
“^Life”	“Life is boring”	Yes	“Life” 패턴이 문자열의 제일 앞 부분에 있다.
“^Life”	“My Life is boring”	No	“Life” 패턴이 문자열의 제일 앞 부분에 없다.

정규 표현식

메타문자: $

- 문자열의 제일 끝 부분과 일치함을 의미.

정규표현식	문자열	Match?	설명
“Python$”	“Python is easy”	No	“Python” 패턴이 문자열의 제일 끝 부분에 없다.
“Python$”	“You need Python”	Yes	“Python” 패턴이 문자열의 제일 끝 부분에 있다.

정규 표현식

메타문자: |

- OR의 의미.

정규표현식	문자열	Match?	설명
“love|hate”	“I love you”	Yes	문자열에 “love” 패턴이 있음.
“love|hate”	“I hate him”	Yes	문자열에 “hate” 패턴이 있음.
“love|hate”	“I like you”	No	문자열에 “love”나 “hate” 패턴이 없음.

정규 표현식

메타문자: ( )

- ( )로 에워싸서 패턴 그룹을 정의할 수 있다.

예).

```
import re
my_regex  = re.compile( "([0-9]+)[^0-9]+([0-9]+)" )
m = my_regex.search("Anna is 15 years old and John is 12 years old.")
print(m.group(0))                    # 전체 .
print(m.group(1))                    # 첫 번째 패턴 그룹.
print(m.group(2))                    # 두 번째 패턴 그룹.
```

- 다음은 위와 동일한 정규 표현식이다:

예). my_regex = re.compile("(□d+)□D+(□d+)")

정규 표현식

메타문자: ( )

- ( )로 에워싸서 패턴 그룹을 정의할 수 있다.

예). “전화 번호 감추기”

```
my_regex = re.compile("(□D+)(□d+)□D+(□d+)□D+(□d+)")
m = my_regex.search("John 010-1234-5678")
print((m.group(1)).strip() + " " + m.group(2) + "-****-****")
```

예). “전화 번호 추출”

```
my_regex = re.compile("(□D+)((□d+)□D+(□d+)□D+(□d+))")
m = my_regex.search("John 010-1234-5678")
print("Phone number : " + m.group(2))                    # ( ) = Group #2.
```

실습 #0201

→ 정규 표현식을 사용해 본다. ←

→ 사용: **ex_0201a.ipynb** ~ **ex_0201c.ipynb** ←

워드 클라우드 (Word Cloud):

- 키워드의 도수에 비례한 폰트 크기.

워드 클라우드 (Word Cloud):

- 다음과 같은 순서로 생성할 수 있다.

  1). 단어 단위로 토큰화 한다.

  2). 정제 및 정규화.

  3). 단어의 도수분포표 작성.

  4). 키워드 추출.

  5). WordCloud()의 인자값 설정 및 출력.

실습 #0202

→ 영문 자료로 Word Cloud를 만들어 본다. ←

→ 사용: **ex_0202.ipynb** ←

실습 #0203

→ 파이썬의 한글 분석 방법에 대해서 알아본다. ←

→ 사용: **ex_0203a_colab.ipynb** , **ex_0203b_colab.ipynb** ←

실습 #0204

→ 한글 자료로 Word Cloud를 만들어 본다. ←

→ 사용: **ex_0204_colab.ipynb** ←

모듈 #2 : 끝

문의:

sychang1@gmail.com